EECS 351-1: Intro to Computer Graphics **Project A: Moving Shapes**
Instructor: Jack Tumblin **Winter 2019**

GOALS:

A) Make a collection of several obviously-different, visually-distinctive rigid 3D '**parts**' stored sequentially on the GPU. Each 'part' is a fixed set of colorful vertices that define a closed 3D shape. Your program creates all these 'parts', transfers them all sequentially to the GPU, just once for storage, and then draws them as needed using 'drawArray()' commands that specify the starting vertex and vertex count. Use any WebGL drawing primitive (*e.g.* points, lines, triangles…) in any combination you wish.

B) Assemble your '3Dparts' to make several kinds of animated, jointed 3D '**objects**'. These objects move independently in visually-interesting ways – they might fly, walk, flex, pivot, dance, swim, or more. Each '**object**' consists of several '**parts**' connected only by fixed or hinged joints. A 'fixed' joint locks one part rigidly to another to form a new rigid part. A 'hinged' joint links one point (or axis) you chose on the surface of one part to another point (or axis) you chose on the surface of the other part, and allows rotation around that point (or axis).
You will build each 'object' by using a push-down stack of matrices (e.g. 'modelMatrix') to construct all the various transformations needed to draw each 'part' of each 'object' you will draw and move on-screen.

C) Interactive Animation: these jointed objects should then move dramatically, smoothly, and continuously without any user input (animation), but also respond to users' mouse & keyboard inputs and GUI controls.

Our goal is to learn enough WebGL to make an *interactive* drawing that *you* find interesting and compelling. You may choose to draw ANYTHING with multiple joints: an octopus? Fractal trees that grow and wave in the wind? An N-legged walking creature whose legs consist of 2 or more segments? (Google/Bing: 'DaintyWalker', 'Strandbeest', 'Catmull hand video', 'Jointed Rigid Objects")? Bicycle? bird? helicopter?

Requirements: **Project Demo Day (and due date): Mon Jan 28, 2019**

A) -- In-Class Demo: on the Project's due date (Mon Jan 28) you will demonstrate your completed program to the class. Two other students will each evaluate your work on a 'Grading Sheet', as may Professor Tumblin and assistants. You then have a few days to revise your project to apply what you learned on Demo Day…

B) -- Submit your finalized project to CMS/Canvas no more than 72 hours later (11:59PM Thurs Jan 31) to avoid late penalties. Submit just one single compressed folder (ZIP file) that contains:
1) your written project report as a PDF file, and
2) one folder that holds sub-folders with all Javascript source code, libraries, HTML, etc. (mimic the 'starter code' ZIP-file organization) We must be able to read your report & run your program in the Chrome browser by simply uncompressing your ZIP file, and double-clicking an HTML file found inside, in the same directory as your project report.

c) -- *IMPORTANT:* Name your ZIP file and the directory inside as: **Familyname**Personalname_ProjA
For example, my project A file would be: TumblinJack_ProjA.zip. It would contain sub-directories such as 'lib' and files such as TumblinJack_ProjA.pdf (a report), TumblinJack_ProjA.html, TumblinJack_ProjA.js ,etc.
---To submit your work, upload your ZIP file to Canvas→Assignments. DO NOT e-mail projects! (ignored!).
---BEWARE! SEVERE LATE PENALTIES! (see Canvas→Assignments, or the Syllabus/Schedule).

Project A consists of:

1)—Report: A short written, illustrated report, submitted as a printable PDF file as part of your final version (not for Demo Day). Length: >1 page, and typically <5 pages, but you should decide how much is sufficient. A complete report consists of these parts:

A) your name, netID (3 letters, ~3 digits: my netID is jet861), and a descriptive title for your project (e.g. Project A: Planetary Gear Transmission, not just 'Project A'

B) a brief 'User's Guide' that explains your goals, and then gives user instructions on how to control the project as it runs. (*e.g.* "A,a,F,f keys rotate outer ring forwards/backwards; S,s,D,d keys rotate inner ring forwards/backwards; webpage text shows velocity in kilometers/hour.") Your classmates should be able to read ONLY this report and easily run and understand your project without your help.

C) a brief, illustrated 'Results' section that shows **at least 4 still pictures** of your program in action (use screen captures; no need for video), with figure captions and text explanations. NOTE --You'll earn extra credit if you include a correct sketch of your program's scene graph (the 'tree of transforms'. Unsure? See lecture notes DualitySceneGraphs--VanDamm07.pdf. Remember: root node is always the CVV; transform nodes get only 1 parent and 1 child (use 'group' nodes if you need more): '**parts**' are always leaf nodes with no children; and *only* group nodes have multiple children).

2)—Your Complete WebGL Program, which must include:

a) User Instructions: When your program runs, it must explain itself to users. How? You decide! Perhaps print a brief set of user instructions below the canvas object? Or print 'press F1 for help'? Create a pop-up window? Perhaps within the 'canvas' element using the 'HUD' method in the book, or in the JavaScript 'console' window (in Google 'Chrome' browser), etc. Your program should never puzzle its users, or require your presence to explain, find, or use any of its features.

b)— At least two (2) different 3D 'parts' that you designed yourself—no copying! The shape you design must be more complex than a rectangle or a cube, and contain 12 or more vertices stored sequentially in Vertex Buffer Object (VBO). In step d) below, you will assemble these parts to make moving, jointed objects.

c)— Smoothly-varying Per-Vertex Colors. All 3D parts must vary their colors between their vertices, using at least 3 obviously-different vertex colors (not just 2!), and interpolated by proper use of 'varying' variables in shaders. Every vertex must have RGB color attributes in the VBO ((see 'multiple attributes' in your book; be sure you understand the proper use of 'stride' and 'offset' as described in Chapter 5 and demonstrated in starter code). Of course, do not use 'canvas' drawing primitives (e.g. context.filledRect()); you must use vertex buffer objects (VBOs) for sets of vertices as demonstrated in WebGL Programming Guide, Chapter 3,4).

d)—At least two (2) different *Kinds* of jointed, moving objects. Each different 'kind' of object must be distinct, unrelated, and able to move entirely independently – a 'kind' of butterfly and a 'kind' of tiger; a 'kind' of steam-shovel and 'kind' of helicopter; but not a 'kind' of robot arm and a 'kind' of robot leg (because both are just parts of the same robot object—the legs will never walk away from the arms). EACH of these 2 (or more) different 'kinds' of objects should:

- each require a differently-shaped 'scene graph' to describe its jointed parts (a different sequence of nodes; a differently-shaped 'tree of transformations'; a different arrangement/topology of joints), and
- get assembled from at least **3 or more 'parts'**, connected at clearly-different hinge-point locations, and
- move fully independently; one 'kind' of object cannot depend on position, size, orientation, etc. CAUTION! Don't use just one 'current angle' variable; independent movements require DIFFERENT angles that vary at DIFFERENT rates and reverse direction at DIFFERENT times and/or angles.
- contain **two or more sequential hinge joints.** 'Sequential' means we have a hinge-joint that, when rotated to a new joint angle, moves a rigid 'part' around its hinge point, and that 1st part in turn has

another, different hinge-point at a different location, where a 2nd hinge joint connects this part to a 2nd part. Changing only the 1st joint-angle will move BOTH the 1st part AND the 2nd part.
Changing only the 2nd joint angle will move ONLY the 2nd part.

- I strongly recommend that you make a separate JS function to draw each 'kind' of object (e.g. drawRobot(), drawHelicopter()) using the current contents of your modelMatrix. You can then easily draw many robots and many helicopters anywhere, at any on-screen size, by just setting the modelMatrix and calling the function to draw the object. I also recommend that your object-drawing functions use hinge-joint angles as arguments (e.g. drawRobot(shoulder,elbow, wrist,fingers)) so that you can easily change angles for every object-drawing made.
- **HINT**: Remember, a well-designed jointed object often requires you to 'push' and 'pop' matrices from your modelView matrix stack as you draw it, as demonstrated in the Week 3 'Stretched Robot' starter code. 'Two sequential joints' means that your scene graph has at least one group node that is the child of the 1st hinge-transform node, and that group node must have at least two child nodes: one that draws the movable 1st part, and another whose descendants will include the 2nd hinge-transform, which rotates around a point fixed elsewhere on the 1st part. That 2nd transform node then has as one of its descendants a node to draw the 2nd part.

  For example, A robot arm-and-hand satisfies this requirement: torso (part 1) attaches to displaced upper arm (part 2) via hinge-like shoulder (joint 1); the upper arm then attaches to displaced lower arm (part 3) through a hinge-like elbow (joint 2); lower arm then attaches to displaced hand (part 4) through a hinge-like wrist joint (joint 3). Torso movement moves all the sequentially attached parts. Conversely, you will *not* satisfy this requirement with a stick-legged starfish. If made of a pentagonal body and 5 hinged but joint-free single-segment arms, it has 6 parts and 5 joints, but no sequential joints. Adjusting one arm joint has no effect on any other part. Its scene graph holds a body transform followed by 5 children; one for each arm-angle transform, and no arm is the descendant of another arm.

Your JavaScript program must create ModelMatrix-like concatenations of 4x4 matrices to transform all the vertices of your object, to position, scale, and orient/rotate them in pleasing ways. Construct your 'model' matrix in Javascript using the **`cuon-matrix-quat03.js`** library supplied in the starter code, then apply it to the contents of vertex-buffer objects (VBOs) by matrix multiplies in your Vertex Shader.

e)—Interactive Animation: Like all projects in this course, your program must show a picture that moves and changes, both by itself (animation) and in response to user inputs (interaction) from mouse or keyboard. Users must be able to pause/unpause the animation, and add easily-visible movements from user controls. Controls may be keyboard, mouse, HTML buttons/sliders/edit boxes, etc.

Test your code to ensure ALL your user controls visually obvious – if I can't see the effect easily after 3-5 repetitions (repeated key-press, or mouse-click, or mouse-drag) then that control is not suitably 'usable' and 'obvious' – and thus it might get overlooked during grading and/or might not earn any credit.

d)—Smooth movement only: As your objects move due to animation and/or user inputs they must travel smoothly, continuously; they must not make any sudden jerky 'jumps' from one position to another.

c)—Event Handlers: your program and its shaders should make proper use of registered event handlers for **keyboard, mouse and display**. You have many choices here, including the simple methods demonstrated in Chapter 3 and in the 'starter code' posted. Event handlers let your programs respond to the mouse, respond to changes in the display window size, respond to keyboard inputs, and more. You are also welcome to use better, external libraries for user-interfaces in HTML/JavaScript, such as basic CSS controls or Google's dat.GUI: http://workshop.chromeexperiments.com/examples/gui/#1--Basic-Usage ; https://code.google.com/p/dat-gui/

3)—Note the opportunities for extra credit by adding more features to your project; see Grading Sheet.

Sources & Plagiarism Rules:

Simple: *never* submit the work of others as your own.
You are welcome to begin with the book's example code and the 'starter code' I supply; you can keep or modify any of it as you wish without citing its source. I strongly encourage you to always start with a basic graphics program (hence 'starter code') that already works correctly, and incrementally improve it; test, correct, and save a new version at each step.

I ***want*** you to explore -- to learn from websites, tutorials and friends anywhere (e.g. GitHub, StackOverflow, MDN, CodeAcademy, OpenGL.org, etc), and to apply what you learn in your Projects. Please share what you find with other students, too -- list the URLs on CMS/Canvas discussion board, etc. and list in the comments the sources of ideas that helped you write your code.

BUT always, ALWAYS credit the works of others— ***no plagiarism!***

Plagiarism rules for writing essays apply equally well to writing software. You would never cut-and-paste paragraphs or whole sentences written by others and submit it as your own writing: and the same is true for whole functions, blocks and statements. ***Take their good ideas, but not their code*** add a gracious comment that recommends the inspiring source of those good ideas, and then write your own, better code in your own better style; stay compact, yet complete, create an easy-to-read, easy-to-understand style.

Don't waste time trying to disguise plagiarized code by rearrangement and renaming (MOSS won't be fooled). Instead, study good code to grasp its best ideas, learn them, and make your own version in your own style. Take the ideas alone, not the code: make sure your comments properly name your sources. (And, ugh, if I find plagiarism evidence, the University requires me to report it to the Dean of Students. Ugly.)

Also, please note that I apply the 'MOSS' system from Stanford (https://theory.stanford.edu/~aiken/moss/) and if I find any plagiarism evidence (sigh), the University requires me to report it to the Dean of Students for investigation. It's a defeat for all involved: when they find misconduct they're very strict and very punitive).